

CROSS-PLATFORM SUBSELECT METADATA EXTRACTION

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates in general to computer-implemented database systems, and, in particular, to a technique for cross-platform subselect metadata extraction.

2. Description of Related Art

Databases are computerized information storage and retrieval systems. A Relational Database Management System (RDBMS) is a database management system (DBMS) which uses relational techniques for storing and retrieving data. Relational databases are organized into tables which consist of rows and columns of data. The rows are formally called tuples or records. A database will typically have many tables and each table will typically have multiple tuples and multiple columns. The tables are typically stored on direct access storage devices (DASD), such as magnetic or optical disk drives for semi-permanent storage.

In a DBMS, data records are stored in table spaces. A table space may contain one or more tables. Each table space contains a number of space map pages. Each space map page covers a number of data pages. One or more records can be stored in a single data page. All data pages within a single table space must have the same page size. Typically, a page contains 4096 bytes.

Some current systems provide a technique for obtaining column names and data types of a table or a query result. For example, the DB2® Universal Database (UDB) system from International Business Machines, Corporation has a DESCRIBE command for use in defining a result set (i.e., a query result). In particular, the DESCRIBE command lists column names and data types of a query result (i.e., a result set). The DB2® UDB system supports the DESCRIBE command for various platforms or operating systems, including UNIX®, Windows NT®, and OS/2® platforms and enables a developer to access table, view,

or query metadata information. Metadata information comprises information about other information. For example, metadata provides type information about columns in a table.

SQLJ is a set of standards used to define how the Java™ programming language can be used with the Structured Query Language (SQL). SQLJ enables developers to use Java™ data types or classes as data types in SQL. Therefore, relational tables accessed within an SQLJ application may contain columns having Java™ data types or classes. A SQLJ stored procedure is considered a specialized type of SQLJ application. Additionally, a SQLJ iterator describes columns for a result set using Java™ types.

On the other hand, some systems do not provide such techniques for obtaining the types of a result set. The DB2® Version 5 of the OS/390® platform, on the other hand, does not have a DESCRIBE command. However, a developer can alter the Data Manipulation Language (DML) statement so that it returns no data, but allows full access to the metadata similar to that provided in the above DESCRIBE command.

The types of a result set is useful and desirable information. There is a need in the art of an improved technique of obtaining this information, across platforms.

SUMMARY OF THE INVENTION

To overcome the limitations in the prior art described above, and to overcome other limitations that will become apparent upon reading and understanding the present specification, the present invention discloses a method, apparatus, and article of manufacture for a computer-implemented technique for cross-platform subselect metadata extraction.

In accordance with the present invention, a metadata extraction system executes a query against a database on a data storage device connected to a computer. Initially, the query is modified to replace one or more selected clauses with a false clause. The modified query with the false clause is executed. Metadata is retrieved from the result set obtained by executing the modified query. The metadata is used to obtain column types, which are converted to Java™ types. Then, a SQLJ iterator is generated, which has parameters for the Java™ types.

BRIEF DESCRIPTION OF THE DRAWINGS

Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. 1 schematically illustrates a hardware environment of an embodiment of the present invention;

FIG. 2 is a hardware environment used to implement a client computer in one embodiment of the invention;

FIG. 3 is a hardware environment used to implement a server computer in one embodiment of the invention;

FIG. 4 illustrates the Stored Procedure Builder topology;

FIG. 5 is a flow diagram of a general overview for building and compiling stored procedures using the present invention; and

FIG. 6 is a flow diagram of the processing performed by the Metadata Extraction System.

DETAILED DESCRIPTION

In the following description of one embodiment of the invention, reference is made to the accompanying drawings which form a part hereof, and which is shown by way of illustration a specific embodiment in which the invention may be practiced. It is to be understood that other embodiments may be utilized as structural changes may be made without departing from the scope of the present invention.

Hardware Environment

FIG. 1 schematically illustrates a hardware environment of an embodiment of the present invention, and more particularly, illustrates a typical distributed computer system using the network 100 to connect client computers 102 executing client applications to a server computer 104 executing software and other computer programs, and to connect the server system 104 to data sources 106.

A typical combination of resources may include client computers 102 that are personal computers or workstations, and a server computer 104 that is a personal computer, workstation, minicomputer, or mainframe. These systems are coupled to one another by various networks, including LANs, WANs, and the Internet. The data sources 106 may be geographically distributed.

A client computer 102 typically executes a client application and is coupled to a server computer 104 executing server software. The client application program is typically a software program which can include, *inter alia*, multi-media based applications, e-mail applications, e-business applications, or workflow applications. The server computer 104 also uses a data source interface and, possibly, other computer programs, for connecting to the data sources 106. The client computer 102 is bi-directionally coupled with the server computer 104 over a line or via a wireless system. In turn, the server computer 104 is bi-directionally coupled with data sources 106. The computer 110 is bidirectionally coupled with the client computers 102 and the server computers 104.

The computer programs executing at each of the computers, including the Secure Access System 110, are comprised of instructions which, when read and executed by the computers, cause the computers to perform the steps necessary to implement and/or use the present invention. Generally, computer programs are tangibly embodied in and/or readable from a device, carrier, or media, such as memory, data storage devices, and/or data communications devices. Under control of an operating system, the computer programs may be loaded from the memory, data storage devices, and/or data communications devices into the memory of each computer for use during actual operations.

Thus, the present invention may be implemented as a method, apparatus, system, or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" (or alternatively, "computer program product") as used herein is intended to encompass a computer program accessible from any computer-readable device, carrier, or media, including the internet. Of course, those skilled in the art will recognize many modifications may be made to this configuration without departing from the scope of the present invention.

Those skilled in the art will recognize that the environment illustrated in FIG. 1 is not intended to limit the present invention. Indeed, those skilled in the art will recognize that other alternative hardware environments may be used without departing from the scope of the present invention. For example, the system described can apply either to a general-purpose computer, or to a special-purpose system.

FIG. 2 is a hardware environment used to implement a client computer in one embodiment of the invention. The present invention is typically implemented using a client computer 200, which generally includes a processor 202, random access memory (RAM) 204, data storage devices 206 (e.g., hard, floppy, and/or CD-ROM disk drives, etc.), data communications devices 208 (e.g., modems, network interfaces, etc.), display devices 210 (e.g., CRT, LCD display, etc.), and input devices 212 (e.g., mouse pointing device, keyboard, and CD-ROM drive). It is envisioned that attached to the client computer 200 may be other devices, such as read only memory (ROM), a video card, bus interface, printers, etc. Those skilled in the art will recognize that any combination of the above components, or any number of different components, peripherals, and other devices, may be used with the client computer 200.

The client computer 200 operates under the control of an operating system (OS) 214. The operating system 214 is booted into the memory 204 of the client computer 200 for execution when the client computer 200 is powered-on or reset. In turn, the operating system 214 then controls the execution of one or more computer programs 216, such as a Stored Procedure Builder 218, by the client computer 200. The Stored Procedure Builder 218 comprises a Metadata Extraction System 220. The present invention is generally implemented in these computer programs 216, which execute under the control of the operating system 214 and cause the client computer 200 to perform the desired functions as described herein.

The operating system 214 and computer programs 216 are comprised of instructions which, when read and executed by the client computer 200, causes the client computer 200 to perform the steps necessary to implement and/or use the present invention. Generally, the

operating system 214 and/or computer programs 216 are tangibly embodied in and/or readable from a device, carrier, or media, such as memory 204, data storage devices 206, and/or data communications devices 208. Under control of the operating system 214, the computer programs 216 may be loaded from the memory 204, data storage devices 206,
5 and/or data communications devices 208 into the memory 204 of the client computer 200 for use during actual operations.

Thus, the present invention may be implemented as a method, apparatus, or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" (or
10 alternatively, "computer program product") as used herein is intended to encompass a computer program accessible from any computer-readable device, carrier, or media. Of course, those skilled in the art will recognize many modifications may be made to this configuration without departing from the scope of the present invention.

Those skilled in the art will recognize that the environment illustrated in FIG. 2 is not
15 intended to limit the present invention. Indeed, those skilled in the art will recognize that other alternative hardware environments may be used without departing from the scope of the present invention.

FIG. 3 is a hardware environment used to implement a server computer in one embodiment of the invention. In the environment, a computer system 302 is comprised of
20 one or more processors connected to one or more data storage devices 304 and 306 that store one or more relational databases, such as a fixed or hard disk drive, a floppy disk drive, a CDROM drive, a tape drive, or other device.

Operators of the computer system 302 use a standard operator interface 308 to transmit electrical signals to and from the computer system 302 that represent commands for
25 performing various search and retrieval functions, termed queries, against the databases.

In one embodiment of the present invention, the RDBMS software comprises the DB2® product for the OS/390® operating systems. Those skilled in the art will recognize,

however, that the present invention has application to any RDBMS software, whether or not the RDBMS software uses SQL.

As illustrated in FIG. 3, the DB2® system for the OS/390® operating system includes three major components: the Internal Resource Lock Manager (IRLM) 310, the Systems Services module 312, and the Database Services module 314. The IRLM 310 handles locking services for the DB2® system, which treats data as a shared resource, thereby allowing any number of users to access the same data simultaneously. Thus concurrency control is required to isolate users and to maintain data integrity. The Systems Services module 312 controls the overall DB2® execution environment, including managing log data sets 306, gathering statistics, handling startup and shutdown, and providing management support.

At the center of the DB2® system is the Database Services module 314. The Database Services module 314 contains several submodules, including the Relational Database System (RDS) 316, the Data Manager 318, the Buffer Manager 320, and other components 322 such as a SQL compiler/interpreter. These submodules support the functions of the SQL language, i.e. definition, access control, interpretation, compilation, database retrieval, and update of user and system data.

The present invention is generally implemented using SQL statements executed under the control of the Database Services module 314. The Database Services module 314 retrieves or receives the SQL statements, wherein the SQL statements are generally stored in a text file on the data storage devices 304 and 306 or are interactively entered into the computer system 302 by an operator sitting at a monitor 326 via operator interface 308. The Database Services module 314 then derives or synthesizes instructions from the SQL statements for execution by the computer system 302.

Generally, the RDBMS software, the SQL statements, and the instructions derived therefrom, are all tangibly embodied in a computer-readable medium, e.g. one or more of the data storage devices 304 and 306. Moreover, the RDBMS software, the SQL statements, and the instructions derived therefrom, are all comprised of instructions which, when read and

executed by the computer system 302, causes the computer system 302 to perform the steps necessary to implement and/or use the present invention. Under control of an operating system, the RDBMS software, the SQL statements, and the instructions derived therefrom, may be loaded from the data storage devices 304 and 306 into a memory of the computer system 302 for use during actual operations.

Thus, the present invention may be implemented as a method, apparatus, or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" (or alternatively, "computer program product") as used herein is intended to encompass a computer program accessible from any computer-readable device, carrier, or media. Of course, those skilled in the art will recognize many modifications may be made to this configuration without departing from the scope of the present invention.

Those skilled in the art will recognize that the environment illustrated in FIG. 3 is not intended to limit the present invention. Indeed, those skilled in the art will recognize that other alternative hardware environments may be used without departing from the scope of the present invention.

Cross-Platform Subselect Metadata Extraction

A. Overview

The present invention provides a Metadata Extraction System 220 that extracts metadata across platforms. In particular, the metadata that is extracted includes data types of columns in a result set. Without knowledge of which platform the Metadata Extraction System 220 is running on, the Metadata Extraction System 220 can determine the metadata of a result set of a SELECT statement without returning or producing any customer data from the statement.

B. Stored Procedure Builder

The Metadata Extraction System 220 is part of a Stored Procedure Builder, which assists developers with the creation of stored procedures. A stored procedure has zero or more parameters. The stored procedure receives parameter values and is executed via SQL statements. Typically, a stored procedure is used to execute a group of SQL statements without user input (other than the initial parameter values).

In one embodiment of the invention, the stored procedure builder is the DB2® Stored Procedure Builder, which is a graphical application that supports the rapid development of stored procedures. The Stored Procedure Builder may be used to create stored procedures, build stored procedures on local and remote servers, modify and rebuild existing stored procedures, and run stored procedures for testing and debugging the execution of installed stored procedures. The Stored Procedure Builder provides a single development environment that supports many different platforms and works in conjunction with many different applications.

FIG. 4 illustrates the Stored Procedure Builder 400 topology. The topology includes development, debugging, and deployment components. As to development components, one of many Development Applications 402 is used to launch the Stored Procedure Builder 400, on one of many different Platforms 404. In particular, the Stored Procedure Builder can be launched as a separate application from the DB2® Universal Database® program group or from any of the following development applications: IBM® VisualAge® for Java™ Version 3.0 or later, Microsoft® Visual C++® Version 5 or later, Microsoft® Visual Basic® Version 5 or later, or IBM® DB2® Control Center. Some platforms include: AIX®, OS/2®, OS/390®, OS/400®, Sun Solaris®, UNIX®, Windows NT®, Windows 95®, Windows 98®, and Windows 2000®.

As to debugger components, a Debugger Daemon 406 and Debugger Client 408 are provided for debugging a stored procedure built with the Stored Procedure Builder 400. As to Deployment components, the Stored Procedure Builder 400 uses a Java™ Database Connectivity (JDBC) application programming interface (API) 410 to communicate with a

WHERE condition results in no rows being returned, while metadata about the result set is available. In particular, even though no rows are returned, the MetaData via a Java.sql.ResultSetMetaData field is still available and used to construct the iterator necessary for the generating SQLJ code where a SELECT statement is returning rows of data.

5 Since SQL is a complex language where there are a minimal set of reserved words, one or more false SQL statements are constructed and loaded into a list of statements to run. In one embodiment, the number of false SQL statements will be equivalent to the number of WHERE clauses. For example, if a SQL statement has a WHERE clause, then one false WHERE clause is generated and used to replace the original WHERE clause and anything
10 following it (e.g., a GROUP BY clause). If a SQL statement has two WHERE clauses, then one false WHERE clause is generated and used to replace both of the WHERE clauses and everything following them and a second false WHERE clause is generated and used to replace the second WHERE clause and everything following it (i.e., this leaves the first WHERE clause in the SQL statement. The first SQL statement with a false WHERE clause
15 to run without experiencing a SQL exception is the one used to extract the MetaData.

The Metadata Extraction System 220 improves performance by removing the ORDER BY clauses, should they exist in the statement. Also, the Metadata Extraction System 220 handles subselect statements (e.g., a WHERE clause within a WHERE clause). In one embodiment, the pseudocode is not applied to SELECT <column name> INTO
20 statements because results are directly loaded into variables.

Additionally, if the SQL statement requires that the application program pass in parameter values at run time (e.g., the parameters could be used in an expression), default values are used by the Metadata Extraction System 220 when executing a SQL statement with one or more false clauses so that the SQL statement can run properly.

25 FIG. 5 is a flow diagram of a general overview for building and compiling stored procedures using the present invention. Note that FIG. 5 describes processing of one SQL statement for illustration only, and one skilled in the art would recognize that multiple statements may be processed and incorporated into a single stored procedure. Additionally,

although the stored procedure builder, precompiler, and compiler are discussed as separate entities, they may be combined in various manners, resulting in fewer separate entities (e.g., the stored procedure builder and precompiler may be combined).

5 In block 500, a stored procedure builder receives a SQL statement from a user and determines whether the SQL statement requires a SQLJ iterator. A SQL statement requires an iterator if it is a SELECT statement, other than a SELECT INTO statement (which selects data into one or more variables). In block 502, if an iterator is required, processing continues with block 504, otherwise, processing continues with block 506.

10 In block 504, the Metadata Extraction System 220 assists the stored procedure builder with building a stored procedure requiring a SQLJ iterator through construction of a SQLJ iterator for the SQL statement and continues to block 508 for precompiling. In block 506, the stored procedure builder builds a stored procedure and continues to block 508 for precompiling. In block 508, a precompiler precompiles the stored procedure. In block 510, a Java™ compiler compiles the precompiled stored procedure.

15 The following provides pseudocode for the Metadata Extraction System 220:

Is there a WHERE clause in the SQL statement?

N Y

1) | + Establish its position in the statement.

| Is it bounded by whitespace on either side?

20 | N Y

| | + Copy SQL statement up until the WHERE clause, and replace
| | the WHERE clause and everything following it with
| | " WHERE 1 = -1 ". Store this statement into a list
| | where items are stored on a first in first out basis.

25 | | If there is a WHERE clause after the one used previously, using
| | the new WHERE clause position go to #1.

| |
| | + increment the position by 1, and look for more WHERE clauses.

| If found, go to #1 using the new WHERE clause position.

|
+ Is there an "ORDER BY" clause in the statement (Removing extra white space
between the "ORDER" and "BY", leaving a single space)?

5 N Y

2) | + Establish its position in the statement.

| Is it bounded by whitespace on either side?

| N Y

10 | | + Copy SQL statement up until the "ORDER BY" clause
| | and replace the "ORDER BY" clause and everything
| | following it with " WHERE 1 = -1 ".
| | Store this statement into a list where its contents ascribe
| | to the first-in-first-out rule.

15 | + Increment the position by 1, and look for more "ORDER BY"
| clauses. If found, go to #2 using the new "ORDER BY" clause
| position.

20 | + Create a SQL statement (and add it to the list described above) by
| copying the current SQL statement and appending " WHERE 1 = -
| 1 " to the end.

FIG. 6 is a flow diagram of the processing performed by the Metadata Extraction
System 220. In block 600, the Metadata Extraction System 220 receives a SQL statement.
In block 602, the Metadata Extraction System 220 processes WHERE and GROUP BY
clauses in the SQL statement to create a list of SQL statements with false clauses. In block
25 604, the Metadata Extraction System 220 executes each statement in list in sequence until
one executes successfully. In block 606, the Metadata Extraction System 220 obtains type
data for columns from the result set of the successfully executed statement. In block 608,

the Metadata Extraction System 220 converts the type data into JAVA™ types. In block 610, the Metadata Extraction System 220 creates a SQLJ iterator with parameters having the JAVA™ types.

D. Examples

5 D.1 Sample SQLJ code with Different Data Types

In the following example, a default table selection is made via, for example, a user interface on the client computer, with the client and server being on the same machine.

Initially, a user enters an initial SQL statement of: SELECT * FROM SYSCAT.PROCEDURES. Then, the Stored Procedure Builder 218 and Metadata Extraction
10 System 220 work in conjunction to produce SQLJ Stored Procedure QUEST.Procedure4, with a SQLJ iterator that tells the RDBMS how to bind a result set. In particular, the Metadata Extraction System 220 generates the SQLJ iterator. Each parameter of the SQLJ iterator is a Java™ type that corresponds to a type in the initial SQL statement. The Stored Procedure Builder generates the import statements and the public class Procedure4 code,
15 which includes the initial SQL statement and a result set parameter "rs" that returns a result set to a calling application.

```
/**
 * SQLJ Stored Procedure QUEST.Procedure4
 */
20 import java.sql.*;          // JDBC classes
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

#sql iterator Procedure4_Cursor1 ( String, String, String, int, String,
                                   short, byte[], String, java.sql.Timestamp,
```

```
String, String, String, String, String, String,
String, String, String, String, String, short,
String, int, String, String );
```

```
public class Procedure4
```

```
5  {
    public static void procedure4 ( ResultSet[] rs ) throws SQLException, Exception
    {
        Procedure4_Cursor1 cursor1 = null;
        #sql cursor1 =
10  {
            SELECT * FROM SYSCAT.PROCEDURES
        };

        rs [0] = cursor1.getResultSet ( );
    }
15 }
```

D.2 Sample SQLJ code with Different Data Types

In the following example, a table with different datatypes is accessed. Again the stored procedure is built after receiving user input.

Initially, a user enters a SELECT statement. Then, the Stored Procedure Builder 218 and Metadata Extraction System 220 work in conjunction to produce SQLJ Stored Procedure QUEST.Procedure5, with a SQLJ iterator that tells the RDBMS how to bind a result set. In particular, the Metadata Extraction System 220 generates the SQLJ iterator. Each parameter of the SQLJ iterator is a Java™ type that corresponds to a type in the initial SQL statement. The Stored Procedure Builder generates the import statements and the public class

25 Procedure5 code, which includes the initial SELECT statement and a result set parameter "rs" that returns a result set to a calling application.

/**

* SQLJ Stored Procedure QUEST.Procedure5

*/

import java.sql.*; // JDBC classes

5 import sqlj.runtime.*;

import sqlj.runtime.ref.*;

#sql iterator Procedure5_Cursor1 (String, String, String, java.math.BigDecimal,
short, int, long, float, double, java.sql.Date,
Java.sql.Time, java.sql.Timestamp);

10 public class Procedure5

{

public static void procedure5 (ResultSet[] rs) throws SQL Exception, Exception

{

Procedure5_Cursor1 cursor1 = null;

15 #sql cursor1 =

{

SELECT

ALLTYPES.CHAR AS CHAR,

ALLTYPES.VARCHAR AS VARCHAR,

20 ALLTYPES.LONGVARCHAR AS LONGVARCHAR,

ALLTYPES.DECIMAL AS DECIMAL,

ALLTYPES.SMALLINT AS SMALLINT,

ALLTYPES.INTEGER AS INTEGER,

ALLTYPES.BIGINT AS BIGINT,

25 ALLTYPES.REAL AS REAL,

ALLTYPES.DOUBLE AS DOUBLE,


```

ALLTYPES.DATE AS DATE,
ALLTYPES.TIME AS TIME,
ALLTYPES.TIMESTAMP AS TIMESTAMP
FROM
5      ALLTYPES

```

```

    }
    rs[0] = cursor1.getResultSet ( );
  }
}

```

10

Once metadata is obtained by executing a SQL statement with a false clause, JDBC to Java™ conversion is performed. In one embodiment, a hash code conversion table is used. A portion of the hash code conversion table used for the conversion that can be seen in Example 2, in which the ALLTYPES table is accessed, follows:

```

15      jdbcJava.put (new Integer(Types.CHAR),  "String");
      jdbcJava.put (new Integer(Types.VARCHAR), "String");
      jdbcJava.put (new Integer(Types.LONGVARCHAR), "String");
      jdbcJava.put (new Integer(Types.NUMERIC), "java.math.BigDecimal");
      jdbcJava.put (new Integer(Types.DECIMAL), "java.math.BigDecimal");
20      jdbcJava.put (new Integer(Types.BIT),  "boolean");
      jdbcJava.put (new Integer(Types.TINYINT), "byte");
      jdbcJava.put (new Integer(Types.SMALLINT), "short");
      jdbcJava.put (new Integer(Types.INTEGER), "int");
      jdbcJava.put (new Integer(Types.BIGINT), "long");
25      jdbcJava.put (new Integer(Types.REAL),  "float");
      jdbcJava.put (new Integer(Types.FLOAT),  "double");
      jdbcJava.put (new Integer(Types.DOUBLE), "double");

```

```

jdbcJava.put (new Integer(Types.BINARY), "byte[ ]");
jdbcJava.put (new Integer(Types.VARBINARY), "byte[ ]");
jdbcJava.put (new Integer(Types.LONGVARBINARY), "byte[ ]");
jdbcJava.put (new Integer(Types.DATE), "java.sql.Date");
5 jdbcJava.put (new Integer(Types.TIME), "java.sql.Time");
jdbcJava.put (new Integer(Types.TIMESTAMP), "java.sql.Timestamp");

```

Each of the Types.xxxx are defined constants in the class Types that is in the package java.sql. The class Types defines constants that are used to identify generic SQL types, called JDBC types.

10 D.3 Sample SQLJ code with Different Data Types

In the following example, a sample SQL statement is used that includes WHERE clauses.

The following tables AB and XY illustrate sample tables before a stored procedure is executed against them. All columns are declared as INTEGER types in the example.

15 The following statement is used to display the contents of table TBLAB: db2 =>
select * from tblab.

TABLE AB	COLA	COLB
	-----	-----
20	1	12
	2	12
	3	13
	4	14

4 record(s) selected.

The following statement is used to display the contents of table TBLXY: db2 =>
select * from tblxy

TABLE XY	COLX	COLY
	-----	-----
5	2	22
	3	23

2 record(s) selected.

The following stored procedure, SQLJ Stored Procedure QUEST.Procedure6, is built with the Stored Procedure Builder 218 in conjunction with the Metadata Extraction System 220 and includes a SQLJ iterator that tells the RDBMS how to bind a result set.

Initially, a user enters the following statement: SELECT COLA FROM TBLAB WHERE COLA > ALL (SELECT COLX FROM TBLXY WHERE COLX < 0). Note that the SELECT COLX statement within the SELECT COLA statement is referred to as a subselect statement. Also, the SELECT COLA statement has two WHERE clauses. Thus, the Metadata Extraction System 220 generates two different SQL statements with false clauses.

```

/**
 * SQLJ Stored Procedure QUEST.Procedure6
 */
20 import java.sql.*;          // JDBC classes
import sqlj.runtime.*;
import sqlj.runtime.ref.*;

```

```
#sql iterator Procudure6_Cursor1 ( int );
```

```
public class Procedure6
```

```
{
```

```
    public static void procedure6 ( ResultSet[] rs ) throws SQLException, Exception
```

```
5      {
```

```
        Procedure6_Cursor1 cursor1 = null;
```

```
        #sql cursor1 =
```

```
        {
```

```
            SELECT COLA FROM TBLAB
```

```
10            WHERE COLA > ALL (SELECT COLX FROM TBLXY
```

```
                                WHERE COLX<0 )
```

```
        };
```

```
        rs[0] = cursor1.getResultSet( );
```

```
    }
```

```
15 }
```

The following table represents the result of executing the stored procedure public class Procedure6:

```
20
```

COLA

1

2

3

```
25
```

4

4 record(s) selected.

Variable	Mean	SD	Min	Max
Age	31.15	4.86	21	41
Gender	1.00	0.00	1	1
Marital status	1.00	0.00	1	1
Education	13.15	1.46	10	16
Income	1.00	0.00	1	1
Religion	1.00	0.00	1	1
Occupation	1.00	0.00	1	1
Health	1.00	0.00	1	1
Smoking	1.00	0.00	1	1
Alcohol	1.00	0.00	1	1
Exercise	1.00	0.00	1	1
Stress	1.00	0.00	1	1
Depression	1.00	0.00	1	1
Loneliness	1.00	0.00	1	1
Life satisfaction	1.00	0.00	1	1
Quality of life	1.00	0.00	1	1
Health-related quality of life	1.00	0.00	1	1
Physical health	1.00	0.00	1	1
Mental health	1.00	0.00	1	1
Social health	1.00	0.00	1	1
Emotional health	1.00	0.00	1	1
Overall health	1.00	0.00	1	1